SYSTEM, METHOD AND COMPUTER PROGRAM PRODUCT TO MAXIMIZE SERVER THROUGHPUT WHILE AVOIDING SERVER OVERLOAD BY CONTROLLING THE RATE OF ESTABLISHING SERVER-SIDE NETWORK CONNECTIONS

Inventors: Jose K. Raphel
                Kailash Kailash
                Michel K. Susai

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001]   The present application is related to pending application Ser. No. 09/188,709, filed November 10, 1998, entitled "Internet Client-Server Multiplexer," incorporated herein by reference in its entirety.

[0002]   The present application is also related to pending application Ser. No. 09/690,437, filed October 18, 2000, entitled "Apparatus, Method and Computer Program Product for Efficiently Pooling Connections Between Clients and Servers," incorporated herein by reference in its entirety.

BACKGROUND OF THE INVENTION

Field of the Invention

[0003]   The present invention relates generally to Internet client-server applications, and more specifically to a way of maximizing server throughput while avoiding server overload by controlling the rate of establishing server-side network connections.

Background Art

[0004]   The importance to the modern economy of rapid information and data exchange cannot be overstated. This explains the exponentially increasing popularity of the Internet. The Internet is a world-wide set of interconnected computer networks that can be used to access a growing amount and variety of information electronically.

[0005]   One method of accessing information on the Internet is known as the World Wide Web (www, or the "web"). The web is a distributed, hypermedia system and functions as a client-server based information presentation system. Information that is intended to be accessible over the web is stored in the form of "pages" on general-purpose computers known as "servers." Computer users can access a web (or HTML) page using general-purpose computers, referred to as "clients," by specifying the uniform resource locator (URL) of the page. Via the URL, the network address of the requested server is determined and the client request for connection is passed to the requested server. FIG. 1 is a network block diagram showing a plurality of clients and servers connected to the Internet.

[0006]   Once the requested server receives the client request for connection, the client and server must typically exchange three packets of information to setup a connection. The number of packets specified above for opening a connection (or specified below for closing a connection) assumes that there is no packet loss in the process of connection establishment. In the event packet loss occurs, then the number of exchanged packets will increase correspondingly. A page typically consists of multiple URL's and in fact it is not uncommon to find websites with 40 or more URL's per page.

[0007]   Once the connection is established, a client sends one or more URL (page) requests to the server, which consists of one or more packets. The server will then send one or more packet responses back to the client. Once a request and response is exchanged from the client and server, both client and server may close their respective connections. The closing of the connection takes a minimum of four additional packets of information exchange. Therefore, there is a significant amount of processing overhead involved in downloading even a single URL for a client where that client does not already have a connection established with the server.

[0008]   Each packet that reaches the server interrupts the server's CPU to move that packet from the Network Interface Card (NIC) into the server's main memory. This process uses up server resources and results in loss of productivity

on the server's CPU. In addition, to establish a connection at the server side the packet needs to be processed by the driver layer, where Ethernet specific information is handled. The driver layer sends the packet to the IP (Internet Protocol) layer for more processing, where all the IP related processing is handled. After this, the packet is passed to TCP (Transmission Control Protocol) layer, where the TCP related information is processed. The TCP layer consumes significant server resources to create a connection table, etc.

[0009]   Most servers incorporate multitasking, which also consumes server resources and therefore may increase server response time. Multitasking, which is well known in the relevant art(s), is the ability to execute more than one task at the same time. Examples of a task include processing a URL or page request in order to service an existing client, establishing a new connection in order to accept new clients (which involves, at a minimum, essentially three tasks as described above), closing a connection to an existing client (which involves, at a minimum, essentially four tasks as described above), etc. In multitasking, one or more processors are switched between multiple tasks so that all tasks appear to progress at the same time. There are at least two basic types of multitasking that are well known to those skilled in the art, including preemptive and cooperative.

[0010]   Whether the operating system of a particular server (including, but not limited to, application servers and database queuing) uses preemptive or cooperative multitasking, the response time to URL (page) requests increases as there are more tasks in the system, including tasks in the form of URL requests from more clients. In addition, the response time to a page request increases as the number of new clients trying to gain access to the server increases within a short period of time. For example, if a surge of new clients attempt to gain access to the server at the same time, then under certain load conditions the server may spend the majority of its processing resources accepting new clients rather than servicing its existing clients. A surge of new clients can be the result of a popular web site attracting many new visitors, a server attack, and so forth. A server

attack happens with one or more malicious users make regular requests that are issued at a very high rate in the attempt to crash a server.

[0011]   Servers are also faced with the unpredictable and erratic nature of internet traffic and the inconsistent arrival of requests over the web. Many factors contribute to the wide variability of web traffic including the popularity of a URL or website, the variations in performance of the multiple points of web infrastructure encountered by a request as it traverses the net, including routers, switches and proxy devices and the overall congestion on the infrastructure over which the traffic is being carried.

[0012]   Servers are designed to do certain things well. Servers are typically general-purpose machines that are optimized for general tasks such as file management, application processing, database processing, and the like. Servers are not optimized to handle switching tasks, such as opening and closing network connections. Under certain load conditions, these tasks can represent a considerable overhead, consuming a large percentage of the server's processing resources, often on the order of twenty percent and sometimes up to fifty percent. This problem is referred to herein as "connection loading."

[0013]   The server may provide to its existing clients unacceptably slow server response time when the server is forced to spend most of its processing resources accepting new clients and therefore not servicing existing clients. In fact, when there is no limit on the amount of clients a server is accepting and/or servicing, often times the result is declining server performance, including server failure or crash and/or the failure to service some or all requests coming to it. Some servers, once they reach processing capacity, may just drop or block a connection request. When the response time for a server is unacceptably slow and/or has a tendency to crash often and/or the client's connection request is blocked or dropped, the owner of the server may lose business. This loss of business is detrimental to anyone seeking to conduct business over the Internet.

BRIEF SUMMARY OF THE INVENTION

[0014]   The present invention is a system, method and computer program product for maximizing server throughput while avoiding server overload by controlling the rate of establishing server-side network connections. The present invention ensures acceptable server response time by monitoring the current response time of a particular server (or set of servers) for its (or their) existing clients and then only allowing a new client to make a connection with a particular server if the server's current response time will remain acceptable. In an embodiment, the present invention is implemented within an interface unit connecting one or more servers to the Internet, which are in turn connected to a plurality of clients.

[0015]   According to an embodiment of the invention, the method includes the steps of opening a connection between a new client and an interface unit; determining whether a free connection is open between the interface unit and a requested server, and if so, then allowing the new client to access information on the requested server via the free connection; determining whether opening a new connection between the interface unit and the requested server would cause the requested server to allocate an unacceptable amount of its processing resources to servicing one or more existing clients (i.e., whether the server is operating beyond a range that is acceptably close to its determined optimal performance), and if so, then buffering the new client. Once the amount of allocated processing resources reaches an acceptable level, then the method includes the steps of allowing the new client to access information on the requested server via either the free connection or the new connection. After serving the requested information, the method includes the steps of closing the connection between the new client and the interface unit while keeping open the free connection and the new connection between the interface unit and the requested server.

[0016]   In an embodiment of the present invention, multiplexed connections are used and reused to regulate the flow of HTTP requests to a server or server farm

rather than blocking or dropping new requests once maximum server capacity is reached.

[0017]   In another embodiment, the present invention uses an interface unit to compute server load (or performance) by considering the number of connections that have been opened with a server, by monitoring changes in server response time and by monitoring changes in the rate at which such response time is changing.   This helps to avoid server overload.

[0018]   One advantage of the present invention is that it guarantees that a server will have processing resources available to serve a response to a client once the client's request has been passed to the appropriate server.

[0019]   Another advantage of the present invention is that it eliminates a significant cause of server crashes whereby too many new clients in a short period of time are trying to gain access to the server.

[0020]   Yet another advantage of the present invention is that it may give preferential treatment to certain clients in order for the preferred clients to more readily gain access to the server and thus generate more business and enable preferential treatment for higher priority customers for the server owner.

[0021]   Another advantage of the present invention is that it helps to protect the server from a server attack.

BRIEF DESCRIPTION OF THE DRAWINGS/FIGURES

[0022]   The features and advantages of the present invention will become more apparent from the detailed description set forth below when taken in conjunction with the drawings in which like reference characters identify corresponding elements throughout and wherein:

[0023]   FIG. 1 is a network block diagram showing a plurality of clients and servers connected to the Internet;

[0024]   FIG. 2 is a network context diagram for an interface unit according to an embodiment of the present invention;

[0025]   FIG. 3A illustrates server performance according to an embodiment of the present invention;

[0026]   FIG. 3B is a time line illustrating how the present invention computes server overload or performance in a nonintrusive way according to an embodiment;

[0027]   FIG. 4 is a flowchart illustrating the high level operation of the present invention according to an embodiment of the present invention;

[0028]   FIG. 5 is a detailed flowchart illustrating the buffering aspect of the present invention according to an embodiment;

[0029]   FIG. 6 is a detailed flowchart illustrating the buffering aspect of the present invention according to another embodiment;

[0030]   FIG. 7 is a flowchart depicting the operation of the present invention in translating client and server requests to achieve connection multiplexing;

[0031]   FIG. 8 is a flowchart depicting one embodiment of the operation of the present invention in determining the current performance of the requested server according to an embodiment; and

[0032]   FIG. 9 depicts an example computer system in which the present invention can be implemented.

DETAILED DESCRIPTION OF THE INVENTION

[0033]   The present invention is a system, method and computer program product for maximizing server throughput, while avoiding server overload, by controlling the rate of establishing server-side network connections.

[0034]   FIG. 2 is a network context diagram for an interface unit 202 according to an embodiment of the present invention. In an embodiment, interface unit 202 is an intelligent network interface card with a CPU inside a server. Interface unit 202 can also be an intelligent box sitting outside the server, in which case it can serve more than one server. Interface unit 202 can also be a load balancer,

bandwidth manager, firewall, proxy-cache, router, switch, computer system, or any other network device that is located between a client and server.

[0035]   Referring to FIG. 2, a plurality of clients C1, C2, C3 are coupled to the Internet. A plurality of servers S1, S2, S3 are coupled to the Internet by interface unit 202. Servers S1, S2, S3 are collectively referred to as a "server farm." In an embodiment of the present invention, all Internet traffic with the server farm passes through interface unit 202. While the present invention is described in terms of the Internet, the concepts described also apply to other types of networks, as will be apparent to one skilled in the relevant art.

[0036]   In an embodiment of the present invention, interface unit 202 relieves servers S1, S2, S3 of much of the processing load caused by repeatedly opening and closing connections to clients by opening one or more connections with each server and maintaining these connections to allow repeated data accesses by clients via the Internet. This technique is referred to herein as "connection pooling." Interface unit 202 also transparently splices connections from servers and clients using a technique referred to herein as "connection multiplexing." In an embodiment of the present invention, multiplexed connections are used and reused to regulate the flow of HTTP requests to a server or server farm rather than blocking or dropping new requests once maximum server capacity is reached. The techniques of "connection pooling" and "connection multiplexing" are described in detail in related pending application Ser. No. 09/188,709, filed November 10, 1998, titled "Internet Client-Server Multiplexer," incorporated herein by reference in its entirety and Ser. No. 09/690,437, filed October 18, 2000, titled "Apparatus, Method and Computer Program Product for Efficiently Pooling Connections Between Clients and Servers," incorporated herein by reference in its entirety.

[0037]   In the present invention, interface unit 202 avoids server overload by regulating the rate (and the increase in the rate) at which TCP connections received by remote clients are delivered to a server or set of servers. The present invention uses interface unit 202 to compute server load (or performance) by

considering one or more of (but is not limited to): the number of connections that have been opened with a server, by monitoring changes in server response time, by monitoring changes in the rate at which such response time is changing, by monitoring the mix of requests pending at the server at any point in time and by monitoring error/overload messages as they are generated by the server. The maximum number of connections to the server that can be maintained without performance degradation or generating server error/overload messages and the rate at which the server can accept new clients while still providing an acceptable response time to existing clients varies both depending on the kind of server infrastructure implemented as well as the type and rate of requests coming in to that server for any given time period.

[0038]   FIG. 3A is a plotted graph illustrating performance or load of a server. FIG. 3A is a graph representing the number of requests per second to the server or server farm (represented by the y axis) and the number of users or clients currently being served by the server (represented by the x axis). Line 302 represents server throughput, line 304 represents current server response time to a client request, and line 306 represents the rate at which the invention opens connections to the server.

[0039]   Point 308 on throughput line 302 illustrates a point on the graph in which the server has reached maximum throughput. Point 310 on line 302 illustrates the server having similar throughput as point 308 (as does all of the points in between point 308 and 310). Server performance, as represented by line 302, reaches a plateau as shown on the graph when the server reaches its maximum capacity for servicing requests and remains level even as users increase as a result of latencies in request delivery made by the users. A feature of the present invention is to keep the server's performance as close as possible to point 308, as compared to point 310, even though points 308 and 310 show similar amounts of throughput. Comparing points 308 and 310, at point 308 the response time is less, the number of users is less and the number of open connections is greater than at point 310. Therefore, it is desirable for a server to be performing as close as possible to point

308. How the present invention ensures that the server's performance remains as close as possible to point 308 will be described with reference to FIG. 4 below.

[0040]   As stated above, all Internet traffic with the server or server farm passes through interface unit 202. The position of the interface unit 202 enables it to compute server load and performance in a nonintrusive way. This can be illustrated with the time line referenced in FIG. 3B. In FIG. 3B, a client 312 first forwards a request that is intercepted by interface unit 202. This is shown by time line 316. Then, as shown by time line 318, interface unit 202 determines when to forward the request from client 312 to a requested server 314. At time line 320, requested server 314 forwards the requested information which is intercepted by interface unit 202. Here, the present invention may simply calculate the difference between time line 318 (when the request was sent to server 404) and time line 320 (when the request was filled) to determine the server response time. The present invention may also consider the number of pending requests sent to the server and how long they have been pending to calculate server response time. In addition, the invention may use error and overload messages that it has received from the server to adjust what the optimal performance or load should be for a particular server. It is important to note optimal server load is determined on a dynamic basis as optimal performance varies through time depending on the type of requests pending on the server at any point in time. The interface unit 202 considers not just overall server performance knowledge, but also the mix of requests presently pending at the server. In any event, the current response time is calculated in a nonintrusive matter since server 404 is not aware of this calculation. Finally, as shown by time line 322, the requested information is forwarded to client 312 by interface unit 202.

[0041]   FIG. 4 is a flowchart illustrating how the present invention ensures that a server's performance remains as close as possible to point 308 (FIG. 3). FIG. 4 incorporates the "connection pooling" and "connection multiplexing" techniques mentioned above. It is important to note that although FIG. 4

illustrates using "connection pooling" and "connection multiplexing," the present invention is not limited to using these techniques.

[0042]   The process in FIG. 4 begins when a client requests access to one of the servers in the server farm (herein referred to as the "requested server") tended by interface unit 202. A connection is opened between interface unit 202 and the requesting client, and interface unit 202 receives the client request to access the requested server, as shown in step 402.

[0043]   Next, interface unit 202 determines the identity of the requested server as shown in step 404. In one embodiment, this is accomplished by examining the destination network address specified by the client request. In another embodiment, this is accomplished by examining the network address and path name specified by the client request.

[0044]   After determining the identity of the server to which the client request should be directed, interface unit 202 utilizes the "connection pooling" technique by determining whether a free connection (that is, one that is not in use) to the requested server is already open, as shown in step 406.

[0045]   One aspect of the present invention is to limit the maximum number of allowable connections to the requested server. As described above, the requested server utilizes processing resources to open a new connection to the requested server in order to accept a new client. The maximum number of allowable connections may be set in several ways. One way is a hard limit configured by the system administrator. Another way is to dynamically determine the number of maximum connections at which the server response time exceeds a predetermined threshold. Another way is by looking at the queue of requests pending at the server (as opposed to requests buffered on the present invention) and comparing it with the maximum capacity of such server queue. Therefore, if there is a free connection in step 406, then the present invention utilizes that connection to service the client. Also discussed below in step 413 and step 414, interface unit 202 buffers the client when there are no free connections available (and the maximum connections are already allocated). Therefore, it is assumed

that if there is a free connection then there are no clients being buffered by interface unit 202 at that time. At this point, control passes to step 418 where the client's request is translated and passed to the requested server, as is more fully described with respect to FIG. 7 below.

[0046]   Alternatively in step 406, if there is no free connection to the requested server, then the present invention determines the current performance of the requested server, as shown in step 408.   It is important to ensure that an acceptable amount of the requested server's processing resources is being used to process existing clients. As explained above with reference to FIG. 3, the present invention ensures that the server's performance is as close as possible to point 308. Optimal server load is determined on a dynamic basis by considering not just overall server performance knowledge, but also on the mix of requests presently pending at the server. If the present invention determines that the amount of processing resources being used to process existing clients is not acceptable, then the present invention prevents another client from gaining access to the requested server. For example, if there is a sudden surge of new clients attempting to gain access to the requested server at the same time, then without the present invention the requested server would spend most of its processing resources servicing the new clients (i.e., opening connections) and not servicing existing clients. As stated above, this can result in unacceptable server response time and/or a server crash,and/or other server performance problems.

[0047]   For illustration purposes only, assume that the present invention has dynamically determined that with a given mix of requests on the server, in order for the requested server to perform within a range that is acceptably close to point 308 (FIG. 3), the requested server should spend ninety (90) percent of its processing resources to service existing clients and ten (10) percent of its processing resources to accept new clients. Therefore, for the requested server the present invention pre-determines that the requested server's optimal percentage to service existing clients is 90%. In other words, when the requested

server is spending 90% of its resources on servicing existing clients, then its performance is within a range that is acceptably close to point 308 (FIG. 3).

[0048] As shown in step 410 of FIG. 4, the present invention next determines whether the determined performance (step 408) is within a range that is acceptably close to an optimal performance (i.e., point 308 in FIG. 3A). How this may be determined is described in more detail below with reference to FIG. 8 below. If the outcome to step 410 is positive, then this indicates to interface unit 202 that the requested server is performing within a range that is acceptably close to point 308 and therefore the requested server can accept a new client without increasing the response time to an unacceptable level. If the outcome to step 410 is positive, then control passes to step 425.

[0049] The present invention must not service the client if there are other clients that have been buffered previously by interface unit 202 that are still waiting to be serviced, as shown in step 425. In step 425, if there are other clients waiting to be serviced, then control passes to step 414 where the client is buffered by interface unit 202. Alternatively, control passes to step 411.

[0050] In step 411, interface unit 202 ensures that a maximum number of connections to the requested server is not exceeded. Here, the maximum number of allowed connections is compared to the current number of connections to the requested server. If the current number of connections is less than or equal to the maximum number of allowed connections, then control passes to step 412 where interface unit 202 may open a new connection to the requested server. Alternatively, if the current number of connections is greater than the maximum number of allowed connections, then interface unit 202 buffers the client until the current number of connections is less than the maximum number of allowed connections, as shown by step 413.

[0051] Alternatively, if the outcome to step 410 is negative, then this indicates to interface unit 202 that the requested server is not performing as closely as desired to point 308. Here it is likely that the requested server is currently spending more of its processing time performing tasks other than servicing

existing clients than allowed. Here, interface unit 202 buffers the client until the current performance is within a range that is acceptably close to the optimal performance and it is the client's turn to gain access to the requested server, as shown in step 414, and as more fully described with respect to FIGs. 5 and 6 below. Interface unit 202 then determines if there is a free connection open to the requested server, as shown in step 416. Interface unit 202 knows to free up a connection when the client utilizing that connections initiates a FIN (finish) command, a RST (reset) command, or via one of the novel ways described in related pending application Ser. No. 09/690,437, filed October 18, 2000, entitled "Apparatus, Method and Computer Program Product for Efficiently Pooling Connections Between Clients and Servers," incorporated herein by reference in its entirety. In all of these scenarios, interface unit 202 waits until it receives one of these commands before it closes the connection between itself and the client and frees up the connection between interface unit 202 and the requested server. Therefore, if there is a free connection, then interface unit 202 utilizes that connection to service the client and control passes to step 418. Alternatively, interface unit 202 ensures that the maximum number of allowed connections to the requested server is not exceeded, as shown in step 411.

[0052]   Interface unit 202 then translates the client request and passes it to the requested server, as shown in step 418, and as more fully described with respect to FIG. 7 below. After server processing, interface unit 202 receives a response from the requested server, as shown in step 420. The server response is translated and passed to the requesting client, as shown in step 422, and described further below. Finally, interface unit 202 closes the connection with the client as shown in step 424, and the flowchart in FIG. 4 ends. However, by utilizing the "connection pooling" and "connection multiplexing" techniques referenced above, the connection between interface unit 202 and the requested server is not disconnected. However, the present invention may close down the connection if it determines that the server is currently overloaded (i.e., current load is greater than the optimal load).

[0053] FIG. 8 is a flowchart depicting one embodiment of the operation of the present invention in determining the current performance of the requested server, as shown in step 408. Note that by determining the performance of the requested server, the present invention is also determining server load. In step 802, the present invention monitors the changes in server response time for one client request to the next.

[0054] Next, the present invention monitors the rate at which the server response time is changing, as shown in step 804.

[0055] Finally, the present invention determines the current performance of the server based on one or more of the following, the monitored response time, the monitored rate at which the server response time is changing, and the number of connections to the server,. The flowchart in FIG. 4 ends at this point.

[0056] FIG. 5 is a flowchart depicting one embodiment of the operation of the present invention in buffering the client, as shown in step 414. Here, interface unit 202 uses a first-in-first-out method (FIFO) to queue the buffered clients. The FIFO method is well known in the relevant art(s) and is not meant to limit the present invention. In step 502 of FIG. 5, interface unit 202 puts the client at the end of the queue. As other clients in the queue get accepted as new clients by the requested server, interface unit 202 moves the client to the front of the queue, as shown in step 504.

[0057] In step 506, interface unit 202 holds the client at the front of the queue until the current performance is within a range that is acceptably close to the optimal performance (i.e., close to point 308 of FIG. 3). At this point the flowchart in FIG. 5 ends.

[0058] FIG. 6 is a flowchart depicting another embodiment of the operation of the present invention in buffering the client, as shown in step 414. Here, the present invention gives preferential treatment to some clients over other clients. A preferred client may be defined by the server and stored by interface unit 202. For example, a server who manages a web site that sells products to retailers may want to give a large chain store preferential treatment to access its web site over

smaller stores with less significant order volumes. One way in which interface unit 202 can assign the appropriate preferred client value to the client is by the client's network address, as shown in step 602. This can simply be a look-up table with network addresses and associated preferred client values provided to interface unit 202 by the requested server and can be based on one or both of the client's internet address or the port address. Other ways in which interface unit 202 may assign the appropriate preferred client value involves information stored in headers related to clients, previous actions of clients, or by cookies related to clients, etc.

[0059]   The client is placed into the queue based on its preferred client value, as shown in step 604. Here, the client is not automatically placed at the end of the queue. In fact, if the client's preferred client value is higher than any of the other clients in the queue, the client may be placed automatically at the front of the queue. The present invention may also factor other variables into adjusting each client's preferred client value once in the queue. Such factors may include how long the client has been in the queue, and so forth.

[0060]   As other clients in the queue get passed by the interface unit 202 to their requested server, interface unit 202 moves the client to the front of the queue, as shown in step 606.

[0061]   In step 608, interface unit 202 holds the client at the front of the queue until the current performance is within a range that is acceptably close to the optimal performance as was determined for the server by the present invention. At this point the flowchart in FIG. 6 ends.

[0062]   FIG. 7 is a flowchart depicting the operation of the present invention in translating client and server requests to achieve connection multiplexing, as shown in steps 418 and 422 (FIG. 4). In an embodiment of the present invention, multiplexed connections are used and reused to regulate the flow of HTTP requests to a server or server farm rather than blocking or dropping new requests once maximum server capacity is reached.

[0063] In an embodiment, the message traffic is in the form of TCP/IP packets, a protocol suite that is well-known in the art. The TCP/IP protocol suite supports many applications, such as Telnet, File Transfer Protocol (FTP), e-mail and Hyper-Text Transfer Protocol (HTTP). The present invention is described in terms of the HTTP protocol. However, the concepts of the present invention apply equally well to other TCP/IP applications, as will be apparent to one skilled in the art after reading this specification.

[0064] Each TCP packet includes a TCP header and an IP header. The IP header includes a 32-bit source IP address and a 32-bit destination IP address. The TCP header includes a 16-bit source port number and a 16-bit destination port number. The source IP address and port number, collectively referred to as the source network address, uniquely identify the source interface of the packet. Likewise, the destination IP address and port number, collectively referred to as the destination network address, uniquely identify the destination interface for the packet. The source and destination network addresses of the packet uniquely identify a connection. The TCP header also includes a 32-bit sequence number and a 32-bit acknowledgment number.

[0065] The TCP portion of the packet is referred to as a segment. A segment includes a TCP header and data. The sequence number identifies the byte in the string of data from the sending TCP to the receiving TCP that the first byte of data in the segment represents. Since every byte that is exchanged is numbered, the acknowledgment number contains the next sequence number that the sender of the acknowledgment expects to receive. This is therefore the sequence number plus one of the last successfully received bytes of data. The checksum covers the TCP segment, i.e., the TCP header and the TCP data. This is a mandatory field that must be calculated and stored by the sender and then verified by the receiver.

[0066] In order to successfully route an inbound packet from a client to the intended server, or to route an outbound packet from a server to a client, interface unit 202 employs a process known as "network address translation." Network address translation is well-known in the art, and is specified by request for

comments (RFC) 1631, which can be found at the URL http://www.safety.net/RF700631.txt.

[0067]   However, in order to seamlessly splice the client and server connections, the present invention also employs the novel translation technique of "connection multiplexing" as described in detail in related pending application Ser. No. 09/188,709, filed November 10, 1998, titled "Internet Client-Server Multiplexer." According to this technique, the present invention translates a packet by modifying its sequence number and acknowledgment number at the TCP protocol level. A significant advantage of this technique is that no application layer interaction is required.

[0068]   Referring to FIG. 7, the network address of the packet is translated, as shown in step 702. In the case of an in-bound packet (that is, a packet received from a client), the source network address of the packet is changed to that of an output port of the interface unit and the destination network address is changed to that of the intended server. In the case of an outbound packet (that is, one received from a server), the source network address is changed from that of the server to that of an output port of the interface unit and the destination address is changed from that of the interface unit to that of the requesting client. The sequence numbers and acknowledgment numbers of the packet are also translated, as shown in steps 704 and 706 and described in detail below. Finally, the packet checksum is recalculated to account for these translations, as shown in step 708.

[0069]   The present invention may be implemented using hardware, software or a combination thereof and may be implemented in a computer system or other processing system. In fact, in one embodiment, the invention is directed toward one or more computer systems capable of carrying out the functionality described herein. An example computer system 900 is shown in FIG. 9. The computer system 900 includes one or more processors, such as processor 904. The processor 904 is connected to a communication bus 906. Various software embodiments are described in terms of this example computer system. After

reading this description, it will become apparent to a person skilled in the relevant art how to implement the invention using other computer systems and/or computer architectures.

[0070]   Computer system 900 also includes a main memory 908, preferably random access memory (RAM) and can also include a secondary memory 910. The secondary memory 1010 can include, for example, a hard disk drive 912 and/or a removable storage drive 914, representing a floppy disk drive, a magnetic tape drive, an optical disk drive, etc. The removable storage drive 914 reads from and/or writes to a removable storage unit 918 in a well known manner. Removable storage unit 918, represents a floppy disk, magnetic tape, optical disk, etc. which is read by and written to by removable storage drive 914. As will be appreciated, the removable storage unit 918 includes a computer usable storage medium having stored therein computer software and/or data.

[0071]   In alternative embodiments, secondary memory 910 may include other similar means for allowing computer programs or other instructions to be loaded into computer system 900. Such means can include, for example, a removable storage unit 922 and an interface 920. Examples of such can include a program cartridge and cartridge interface (such as that found in video game devices), a removable memory chip (such as an EPROM, or PROM) and associated socket, and other removable storage units 922 and interfaces 920 which allow software and data to be transferred from the removable storage unit 918 to computer system 900.

[0072]   Computer system 900 can also include a communications interface 924. Communications interface 924 allows software and data to be transferred between computer system 900 and external devices. Examples of communications interface 924 can include a modem, a network interface (such as an Ethernet card), a communications port, a PCMCIA slot and card, etc. Software and data transferred via communications interface 924 are in the form of signals which can be electronic, electromagnetic, optical or other signals capable of being received by communications interface 924. These signals 926 are provided to

communications interface via a channel 928. This channel 928 carries signals 926 and can be implemented using wire or cable, fiber optics, a phone line, a cellular phone link, an RF link and other communications channels.

[0073]   In this document, the terms "computer program medium" and "computer usable medium" are used to generally refer to media such as removable storage device 918, a hard disk installed in hard disk drive 912and signals 926. These computer program products are means for providing software to computer system 900.

[0074]   Computer programs (also called computer control logic) are stored in main memory 908 and/or secondary memory 910. Computer programs can also be received via communications interface 924. Such computer programs, when executed, enable the computer system 900 to perform the features of the present invention as discussed herein. In particular, the computer programs, when executed, enable the processor 904 to perform the features of the present invention. Accordingly, such computer programs represent controllers of the computer system 900.

[0075]   In an embodiment where the invention is implemented using software, the software may be stored in a computer program product and loaded into computer system 900 using removable storage drive 914, hard drive 912 or communications interface 924. The control logic (software), when executed by the processor 904, causes the processor 904 to perform the functions of the invention as described herein.

[0076]   In another embodiment, the invention is implemented primarily in hardware using, for example, hardware components such as application specific integrated circuits (ASICs). Implementation of the hardware state machine so as to perform the functions described herein will be apparent to persons skilled in the relevant art(s). In yet another embodiment, the invention is implemented using a combination of both hardware and software.

[0077]   The present invention is described specifically when implemented within an interface unit, such as interface unit 202, that is connected to servers in a farm

for the purpose of offloading connection processing overhead from the servers. However, the present invention can also be applied within other kinds of devices that are in the network connection path between the client and the servers. As network traffic flows through such devices, they all have the opportunity to apply the present invention to offload connection processing. Some examples of such devices are:

- Load Balancers which distribute client network connections between a set of servers in a server farm (local or geographically distributed). The invention can readily be combined with the load balancing function.

- Bandwidth managers which monitor network traffic and meter packet flow. These devices can also use the present invention.

- Firewalls monitor packets and allow only the authorized packets to flow through. The present invention can be used to provide an additional feature within firewalls.

- Routers and switches also lie in the path of the network traffic.

[0078]   The industry trend is to integrate additional functionality (such as load balancing, bandwidth management and firewall functionality) within these devices. Hence, the present invention can easily be incorporated into a router.

[0079]   The specific integration of the present invention into each one of the above devices is implementation specific.

[0080]   The present invention can also be applied within computer systems which are the end points of network connections. In this case, add-on cards can be used to implement the invention and thus offload the main processing elements within the computer system.

Conclusion

[0081] The previous description of the preferred embodiments is provided to enable any person skilled in the art to make or use the present invention. The various modifications to these embodiments will be readily apparent to those skilled in the art and the generic principles defined herein may be applied to other embodiments without the use of the inventive faculty. Thus, the present invention is not intended to be limited to the embodiments shown herein but is to be accorded the widest scope consistent with the principles and novel features disclosed herein.